Security Concepts, Challenges, and Design Considerations for Web Services Integration

Gunnar Peterson, Cigital, Inc. [vita[3]]

Howard Lipson, Software Engineering Institute [vita[6]]

2006-12-14 L2 / L, M[7]

Security risks are inherent in all integration technologies. The emergence of web services as an integration pattern presents new threats and opportunities for a system's security. Beyond the initial hype, where web services were viewed as a security pandemic, lie both real risks and new security paradigms.

Acknowledgements

Reviews by Patrick Christiansen, Pamela Curtis, Bob Ellison, Andy Gordon, Patrick Harding, Gary McGraw, Nancy Mead, Tony Nadalin, Eric Newcomer, Mark O'Neill, and Brian Roddy are gratefully acknowledged. Errors and omissions are the authors'.

Overview

Web services evolved after object-oriented programming and component programming models were already in place, but web services represent a fundamentally different approach based on a document-oriented model designed for interoperability at a document, typically XML, level. Hence, security and software architects must consider message schemas, types, values, and message exchange patterns in their designs. Standards are increasingly important because web services can traverse organizational, geographical, and technical boundaries. Since message exchange is a core part of web services' architectural design, a high level of security must be built into the messages from the outset, as well as into the services and systems.

Protecting the messages that the services and systems operate on is a central aspect of web services security and will be a major focus of this document. Unfortunately, this is not the only security issue that web services developers must be concerned with, and so guidance on other issues will be presented as well. For example, issues of trust relating to services and systems that are not in your direct control pervade the web services landscape and must be addressed early in the development life cycle through security policies and building in a monitoring capability for security violations. The central theme of this article is that web services developers must address security concerns as early as possible in the system development life cycle so as to "build security in" rather than attempt the often futile task of patching security onto a system after security problems are manifested in the field.

Why Develop or Use Web Services?

Before we examine the security challenges and solutions associated with web services, it's important to briefly review the business and organizational needs that led to the development of this emerging paradigm for systems integration.

To support greater business efficiency and agility, information systems and their operations have become increasingly decentralized and, for a variety of historical, technical and business reasons, increasingly heterogeneous. Business processes are distributed among far-flung business divisions, suppliers, partners, and customers, with each participant having their own special needs for technology and automation. As a consequence, the demand for a high degree of interoperability among disparate information systems has never been greater. Moreover, it's critical for this high degree of interoperability to be sustained in the face

3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/200-BSI.html (Peterson, Gunnar)
6. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/15-BSI.html (Lipson, Howard F.)

of rapid evolution of the cooperating systems, as participants continually modify their systems in response to new or changing business requirements.

Traditional assembly and integration methods (and the resulting integration software market stimulated by these methods) are not particularly well suited to this new business environment. These methods rely on a tight coupling between cooperating systems, which requires either the universal deployment of homogeneous systems (unlikely, considering the diversity and broad scale of modern business services) or extraordinarily close coordination among participating development organizations during initial development and sustainment (for example, to ensure that any changes to APIs or protocols are simultaneously reflected in all of the deployed systems). In this business environment, such tight coordination is often impractical (e.g., prohibitively expensive), and rapid evolution in response to a new business opportunity is typically out of the question.

In contrast to traditional assembly and integration methods, web services technology provides a paradigm that uses *messages* (in the form of XML documents) passed among diverse, loosely coupled systems as the focal point for integration. These systems are no longer viewed solely as components in a larger system of systems but also as providers of *services* that are applied to the messages. Web services are a special case of the more general notion of Service-Oriented Architectures (SOA). Service-Oriented Architectures represent (i.e., model) interconnected systems or components as collections of cooperating services. The goal of web services technology is to dramatically reduce the interoperability issues that would otherwise arise when integrating disparate systems using traditional means.

What Are Web Services?

Web services offer a way for programmers and vendors to provide integration points with their systems through the use of synchronous and asynchronous message exchanges. Web services technology is widely available with implementations in many of the most widely used programming languages, such as Java/J2EE, C++, C#, Perl, and Python. Vendors providing prepackaged business applications (such as enterprise resource planning, customer relationship management, and databases) and vendors of other COTS applications typically offer web services integration capabilities to extend their systems' data and functionality across platforms.

Web services retain some of the features of object oriented and component programming models (for example, reuse is a core value for both object oriented programming and web services). Likewise, network distribution of programs is a core element in both component programming and web services. But while retaining and extending many similar goals as object-oriented and component programming, web services is a fundamentally different programming model—instead of components being composed into subsystem or systems, services are composed into higher order services. Web services is an architectural and programming model that achieves interoperability and reusability in the following ways:

- Loosely coupled systems—Service requesters and service providers agree upon an interface and abstract away the implementation details. The integration point is defined by the interface contract, which isolates the participants from the effects of change over time.
- Virtualization services and open standards—Allowing applications to run in a virtual environment based on open standards, independent of the specific details of the underlying operating system or hardware platform, allows for Internet-level scalability and distribution similar to HTTP-based web applications.
- Document-orientation—Interoperability across diverse technology is achieved at runtime by leveraging XML documents as a common way to express and exchange data.

Decoupling systems, virtualization and open standards, and interoperability at the document level are all based on the notion of a service:

> "A service is a location on the network that has a machine-readable description of the messages it receives and optionally returns. A service is therefore defined in terms of the message exchange patterns it supports. A schema for the data contained in the message is used as the main part of the contract established between the service requester and a service provider. Other items of metadata

describe the network address for the service, the operations it supports and its requirements for reliability, security, and transactionality" [Newcomer 2004[8]].

At the core of web services technology is a set of standards:

- Interface definition—*Web Services Description Language (WSDL)* defines the interface for the service methods, schema, and bindings [Chinnici 2006[9]].
- Service Registry—*Universal Description, Discovery and Integration (UDDI)* provides basic registry services to locate services on a network. In practice UDDI is typically augmented by vendor-specific registry functionality for governance and additional registry services [OASIS 2006c[10]].
- Interoperable Protocol—*SOAP* is an XML messaging protocol that works over a variety of transports, including HTTP, TCP, FTP, IIOP, JMS, and SMTP [Gudgin 2003[11], W3C 2006[12]]. SOAP originally stood for "Simple Object Access Protocol," but today SOAP is no longer considered to be an acronym.
  - "SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics." [From the Introduction to *SOAP Version 1.2 Part 1: Messaging Framework*, Gudgin 2003[13].]
- Message Schema—XML provides the document format for exchanging messages [W3C 2006b[14]].

Note that SOAP by itself does not contain security specifications. Moreover, SOAP applications have the ability to essentially bypass network firewalls, which are among the most widely deployed security devices, by using ports that are typically open, such as port 80. Since SOAP itself does not contain security mechanisms, this job is left to other mechanisms such as those described in the WS-Security and WS-Trust specifications. Additionally, network security mechanisms that are used in standard practice, like network firewalls, should be re-examined from a risk management standpoint to understand what protection and security services they deliver (or don't deliver) as part of a web services architecture.

Any technology system that deploys these standards may participate in a web services architecture. Decoupling at various points, (e.g. interfaces may be implemented in a variety of technologies, and SOAP may be used over a variety of transports), allows for both interoperability and heterogeneity.

SOAP & REST-style web services

There are two main types of web services widely deployed today. SOAP web services represent a set of related standards including SOAP, XML, WS-*, and UDDI that work together to provide interoperability. The WS-* standards contain a number of security standards for web services, including WS-Security, WS-Trust, and WS-SecureConversation. In this paper, we focus on SOAP web services and their related security functionality. Unless we specifically indicate otherwise, use of the term web services in this paper implies SOAP web services.

REST-style web services leverage the existing Web infrastructure, that is, plain XML (i.e., without SOAP) over HTTP. Due to the security limitations of basic XML over HTTP, the WS-* framework associated with SOAP Web Services provides security services to fill the gap. REST [Fielding 2000[15]] is more of a grassroots movement, but does not provide security standards, so programmers must design security mechanisms and achieve interoperability without standards support. This ad hoc approach makes it much more of a challenge to "build security in" from the outset. For example, HMACs may be used in REST to authenticate messages, but the programmers are not guaranteed that the formats are interoperable with other systems that may also be networked with the application.

8. #dsy639-BSI_Newcomer 2004

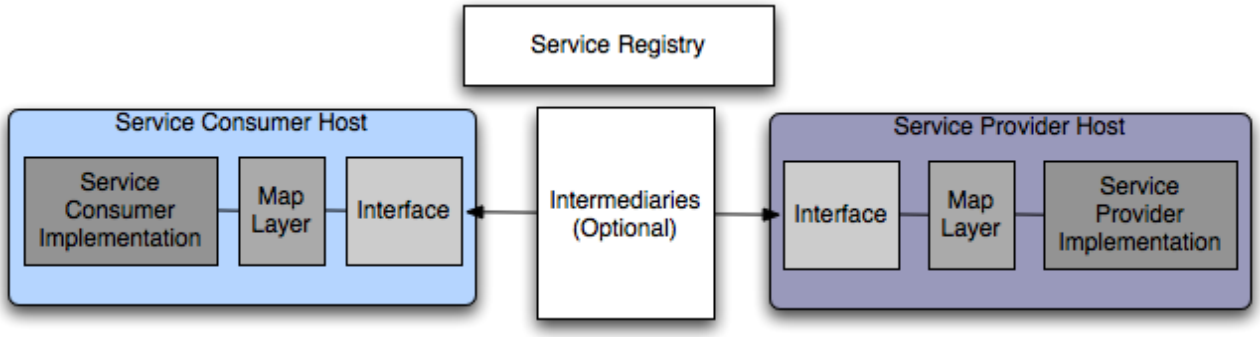
SOAP Web Services Usage Scenarios

SOAP web services are typically implemented in either a synchronous RPC-style or in an asynchronous manner. Each approach has unique considerations.

- Synchronous RPC-style integration is used when information is needed immediately. However, there are typically better methods to acquire data synchronously than SOAP web services, which for example lack robust error handling mechanisms. So with today's technology, this integration approach should not be the default consideration.
- Asynchronous messaging may be used to exchange documents among service requesters and service providers. Asynchronous messaging has the added benefit of allowing for the addition of reliable messaging into a design. HTTP alone lacks the ability to provide guaranteed message delivery. SOAP web services standards, such as WS-ReliableMessaging [Iwasa 2004[16]] provide a way to deliver reliable messaging, like guaranteed delivery and guaranteed order, over a variety of transports, including HTTP.

In practice, a SOAP web services scenario typically comprises a number of participants, including service requesters, service providers, a registry, and a number of intermediaries (such as messaging systems, management systems, metrics and monitoring tools, and even security tools).

Figure 1. Basic elements in a web services architecture



Bringing It All Together: Enterprise Service Bus pattern

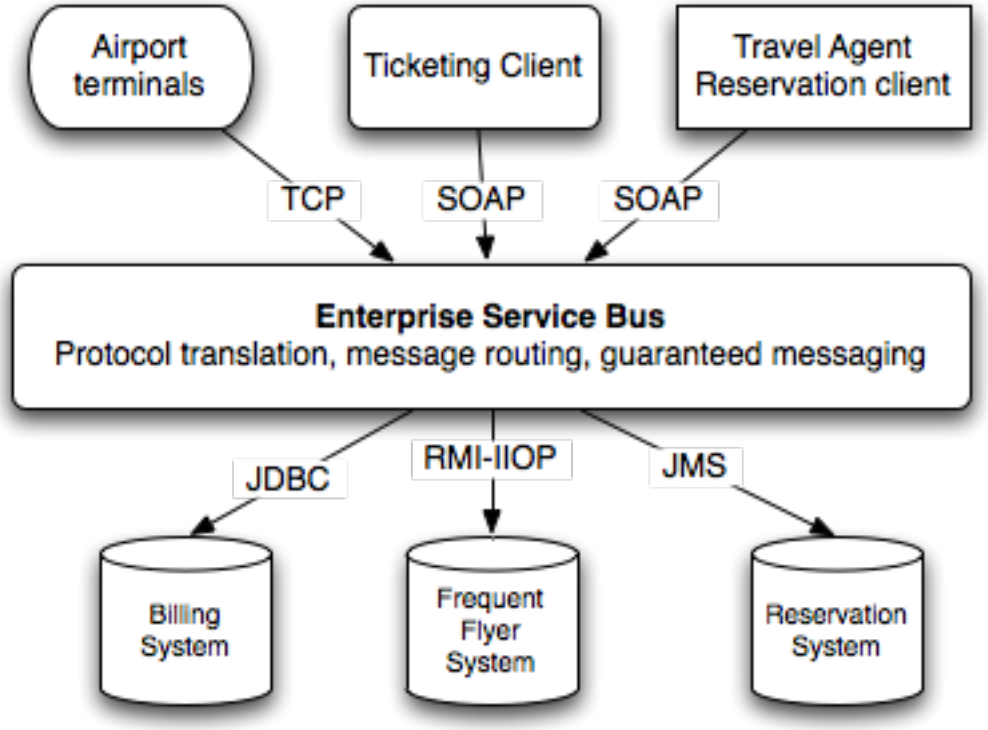
The *enterprise service bus* (ESB) pattern translates communication protocols, aggregates services, and provides a (typically message-oriented) middleware architecture for systems integration [Krafzig 2005[17]]. The enterprise service bus pattern builds on the core web services standards and shows a concrete integration example of web services technologies working in practice.

Brief Industry Example

In this example, an airline reservation system implements the enterprise service bus pattern to integrate servers and terminals across geographic locations. Two client systems communicate to the bus via SOAP web services, while the airport terminal uses TCP sockets. The back end resources are a combination of technologies and protocols. The enterprise service bus is responsible for connectivity to these systems, ensuring that the data is delivered and that messages are communicated in accordance with policy and schema. Note that an enterprise service bus pattern that serves as both a service provider and a service requester engenders many of the same security risks as any web services intermediary.

Figure 2. An example enterprise service bus providing connectivity across platforms

17. #dsy639-BSI_Krafzig 2005



In this example, the enterprise services bus plays a crucial role in integrating a set of collaborative systems that need to interoperate with a high degree of effectiveness to fulfill an overall business mission. Including the ESB, there are seven systems represented; each system may have its own system-specific users, policies, and security technologies, yet the systems are required to interoperate in a secure manner. The security implications, traps, and pitfalls of this integration pattern correspond to the key security issues for SOAP web services. We'll next take a closer look at those issues.

What Are the Key Security Issues for SOAP Web Services?

Consistent with the physical world of entering secure facilities such as airports and military locations, integration points (such as those that web services provide) at these technical and organizational border crossing points are of particular security concern. SOAP web services have two main risk factors:

1. Distributed systems risks: Risks to the services themselves similar to risks that exist in web applications and component applications, such as malicious input attacks like SQL Injection. These risks arise from being distributed on a network. Note that standard IT security controls like network firewalls (which examine only a packet's header) are largely blind to web services risks due to the fact that web services are deployed on commonly available open ports. On the other hand, some types of application firewalls have the ability to examine content, such XML message bodies, and can use application-specific knowledge to thwart some attacks, but are by no means a panacea [Lipson 2005[18]].
2. Message risks: Risks to the document and data that is exchanged among the service requesters and providers. The document may participate in a multi-hop transaction or be subject to inspection by a variety of intermediaries, each operating in different security zones, including separate policy, geographic, technical, and organizational domains. The message's payload may also, of course, contain sensitive data.

Note that many architectures do not use SOAP, but instead just use plain XML (on its own) over HTTP. However, the typical distributed systems risks and message risks described above still apply. With these two high-level risk factors in mind, let's examine how threats and vulnerabilities arise in a web services system.

Threat Examples

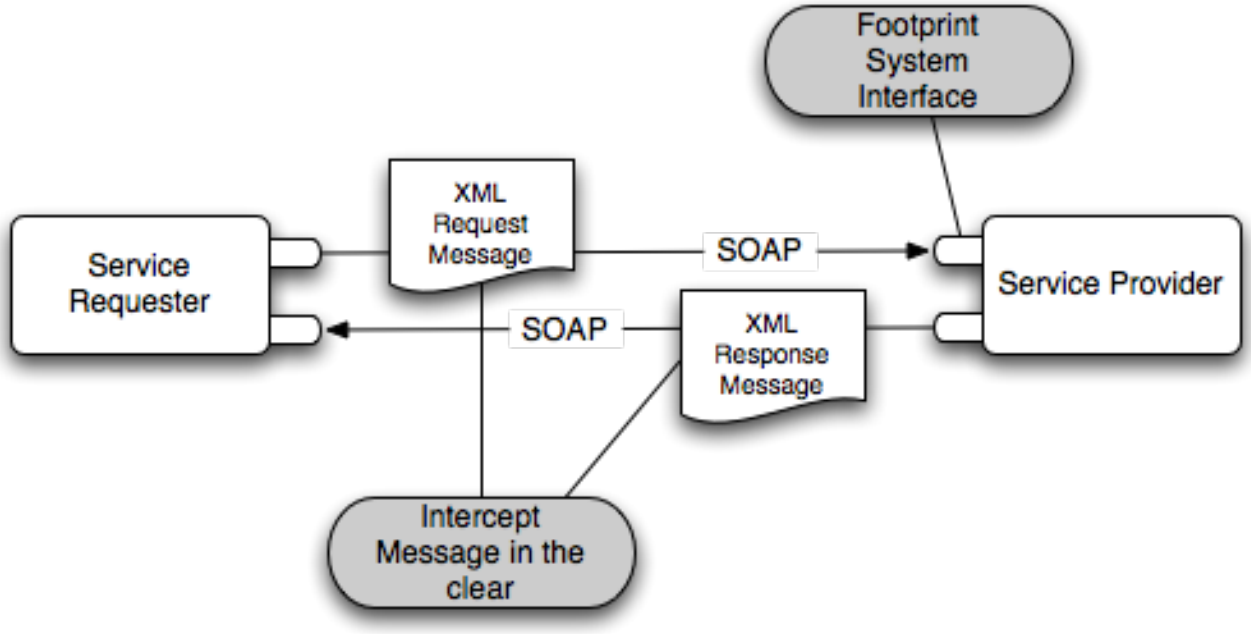
This section describes illustrative, not canonical, threats and vulnerabilities that SOAP web services applications face, using Shirey's model [Shirey 1994[19]], which categorizes threats as disclosure, deception, disruption, and usurpation. The threats are further categorized by service-level threats that are common to most distributed systems and message-level threats that impact SOAP web services XML messages.

Disclosure Threats

Here are some examples of disclosure threats:

- Service-level disclosure: An attacker may footprint a system's data types and operations based on information stored in WSDL, since the WSDL may be published without a high degree of security. For example, in a world-readable registry, the method's interface is exposed. WSDL is the interface to the web services. WSDL contains the message exchange pattern, types, values, methods, and parameters that are available to the service requester. An attacker may use this information to gain knowledge about the system and to craft attacks against the service directly and the system in general.
- Message-level disclosure: XML documents are passed in the clear by default in many implementations. The XML request and response messages posted in SOAP may be passed in the clear, leaving the messages and any data they contain vulnerable to deliberate interception and to inadvertent leakage, such as in audit logs. An asynchronous message system, such as an enterprise service bus, may disclose the content of XML documents, because the message may be cached in the clear. In the industry example, the enterprise service bus administrator may have access to cached documents (if sent in the clear) to and from all six systems. Message disclosure may lead to replay attacks and identity spoofing. The XML schema is another target for disclosure threats.

Figure 3. Example of disclosure threats



Dealing with disclosure threats

At the service level, authentication and authorization mechanisms for service requesters may be used to protect WSDL and related service metadata from disclosure threats. This entails using identity and access management so that the service requester and service provider, which may reside in different namespaces, can share a common security policy, accounts or tokens, and methods, and that the service requester and provider can recognize and understand each other's security tokens and assertions. WSDL describes the

19. #dsy639-BSI_Shirey 1994

service's service interface, ports, and messages. The challenge from a security architecture standpoint is in unifying these concerns into a cohesive security architecture.

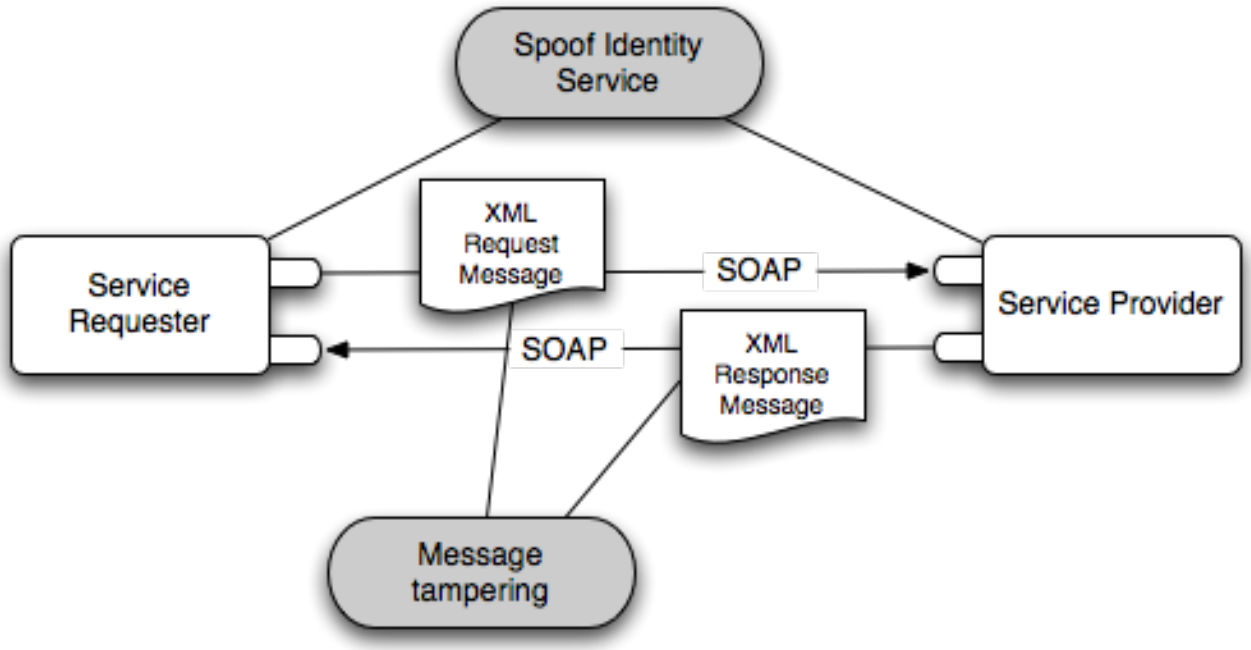
At the message level, there is not a centralized access control mechanism that can protect the XML request and response messages, once they are transmitted. This is especially true in a distributed integration system such as an enterprise service bus. A message generated by a service requester that is posted to the enterprise service bus may traverse the enterprise service bus policy domain and additional systems that are integrated by the enterprise service bus that the initial service requester has no visibility to. Hence, the request (in the form of an XML message) must itself be protected against disclosure threats through a variety of message-level security mechanisms such as encryption and digital signatures. These are discussed in more detail in the next section.

Deception Threats

Here are some examples of deception threats:

- Service-level deception: An attacker may spoof the identity of the service requester, taking a well-formed SOAP XML request message and posting it to the service provider, causing the service provider to assume the response is sent to a valid service requester. Likewise, an attacker may spoof the identity of the service provider to deceive legitimate service requesters to post messages to the spoofed service provider.
- Message-level deception: XML messages are passed without integrity protection by default, leaving the messages vulnerable to tampering. An attacker may tamper with the XML message to execute code and/or gain privileges and information on service requesters and providers. Message tampering may result in injection attacks using XML messages to transport attack requests to other parts of the service infrastructure.

Figure 4. Example of deception threats



Dealing with deception threats

At the service level, spoofing may be mitigated by integrity and authentication mechanisms. Web services supports HTTP authentication methods such as HTTP Basic and mutual authentication through SSL, which provides limited protection against spoofing in point-to-point scenarios. SOAP headers may be used for end-to-end authentication through WS-Security tokens like SAML Assertions, which provide authentication assertions that may be validated by relying parties, and through WS-Security, which defines how to associate

SAML, X.509, Kerberos, and username/password security tokens with a SOAP message header for end-to-end authentication purposes.

At the message level, message integrity through digital signatures and message origin authentication provide a countermeasure against message tampering. XML Signature is the standard used by both SAML and WS-Security for digital signatures. The service may sign all or part of a message. The section “Message-Level Security Standards” below goes into detail about using digital signatures with SOAP messages.

Disruption Threats

These are some examples of disruption threats:

- Service-level disruption: Similar to web application style denial-of-service (DoS) attacks, an attacker may execute denial of service at network level against a web service. Given the range of protocols that may be supported by web services systems, there may be a variety of denial-of-service vulnerabilities.
- Message-level disruption: Since web services are a combination of a variety of technologies (e.g., SOAP, HTTP, and XML), they are vulnerable to combinations of attacks. In other words, an attacker may compose attacks based on vectors in different technologies. In XML Denial of Service (XDoS), the attacker may send an XML message to the parser that forces the XML parser into an infinite recursion and consumes all available computing resources. External references that may be present through non-hardened XML schemas, or poisoned schemas, in XML, DTD, XSD, and WSDL may result in an attacker being able to use XML messages to transport denial-of-service attacks. Additionally, SOAP is geared to provide Internet-level scalability and interoperability for applications, but legacy applications may not be able to handle the increased load. Hence, the increased utilization even by authorized service requesters may yield a service disruption.

Dealing with disruption threats

At the service level, web services denial of service attacks are dealt with in a similar fashion to web application denial of service. Routers, bandwidth monitoring, and other hardware are used to identify and protect against service disruption.

At the message level, it is a tricky proposition for the software security architect, because the XML parser is used to validate the XML message, and the XML parser is the target of this particular attack. One of the main targets of XDoS is DTDs, so web services applications should never use DTDs—DTDs can contain an XDoS attack in a single message. DTDs are vulnerable to infinite recursion attacks that lead to XDoS, and they are known to be vulnerable to other attacks as well. XML parsers may be flooded with very large messages to disrupt the service and/or attacked with very large numbers of small messages. Web services may be throttled to deal with disruption and XDoS attacks, and message size and frequency may be used to assess processing order and execution before parsing begins to deal with this attack.

Note that countermeasures may themselves introduce new security risks. For example, adding encryption and digital signatures to messages increases the processing overhead, which may be exploited by adversaries to cause a denial of service.

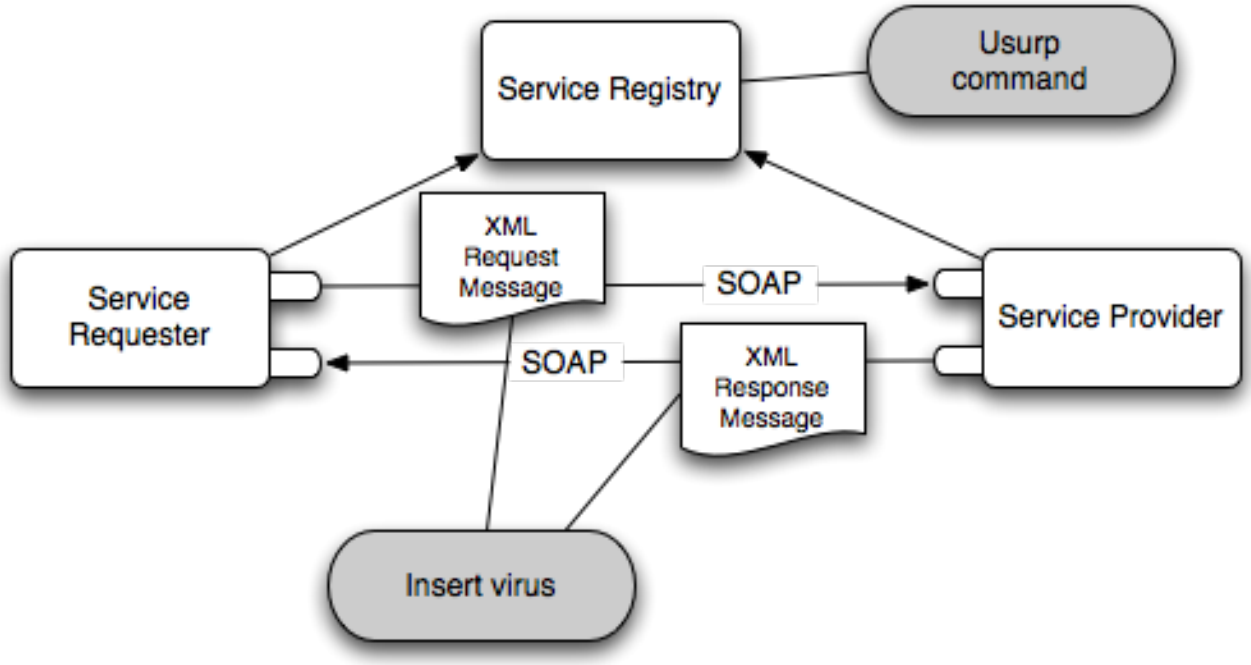
Usurpation Threats

These are some examples of usurpation threats:

- Service-level usurpation: An attacker may usurp command of a system through elevating privileges, for example on the service registry, which contains the policies for the services. In the diagram below, the service requester and service provider publish service metadata to a service registry. The service metadata may contain service policy, addressing and location, quality of service, and interface information. The attacker may exploit the service registry to redirect service requests, change policy, and perform other privileged operations.
- Message-level usurpation: SOAP XML messages may be used to propagate viruses that contain malicious code to steal data. XML messages may be used as a vector to transmit viruses that usurp command through shells or other mechanisms throughout the system. Injection attacks such as SQL

Injection, LDAP Injection, XPath Injection, and XQuery Injection may be used to usurp privileges, drop and alter tables, edit user privileges, and alter schema information.

Figure 5. Example of usurpation threats



Dealing with usurpation threats

At the service level, when service registries are used in web services, they become a central organizing point for a large amount of sensitive information about services. Moreover, some service registries are used at runtime to bind service requesters and service providers dynamically. The service registry (and communication to and from the service registry) should be hardened to the highest degree of assurance that is feasible in the system. For example, service requesters should not have privileges to write data to a service provider entry in the service registry. In a Service-Oriented Architecture, service requesters accessing the service registry is analogous to processes making system calls to privileged modules in a typical operating systems environment, due to the registry's power (like the kernel) to impact policy, runtime, and locale for other services.

At the message level, vendors are beginning to realize the impact that viruses, attached and posted with XML documents, may pose to the environment. For systems that may have XML or binary attachments, virus protection services should be deployed to scan XML and binary messages for viruses in a similar fashion to email messages, that is before the messages are executed for normal business operations.

In addition to the above known threats and as mentioned above, SOAP web services inherit threats and vulnerabilities that are present in its dependent infrastructure, which may include HTTP, TCP, FTP, XML, and other constituent protocols and standards. An integration truism aside from web services is that integration always presents risk when systems are integrated with other systems in ways that extend beyond their original design and purpose.

Web Services Security Standards

Web services use open standards to address some of the new security risks introduced by this emerging paradigm. Open standards have the advantage of interoperability in many cases across technologies, and market forces may deliver best of breed niche products in areas where the point of greatest security pain exists. For example, consumer authentication has been a particular area of concern for financial institutions that perform business online. Advancements in consumer authentication techniques are able to plug into web servers and other existing infrastructure through open standards. Likewise, the SOAP standard is an open

protocol format, designed for composition, so the XML representation of the SOAP message may expand in the future to contain additional security token types and values as new advancements and combinations are developed.

Message-Level Security Standards

The examples in the previous section show the limitations of following a strictly service-based security model in web services. Many existing software security systems rely on authenticating a user to a server, application, or services, and ignore the security issues in the message payload, not addressing key threats to messages in web services. The messages in web services require special protection, since their life cycle may extend beyond the span of control of a given access control system. For example, messages may participate in multihop transactions, be subject to numerous intermediaries, and be routed by enterprise service buses to systems with differing policies and controls. These messages may also be *persisted* (i.e., placed in persistent storage for some period of time) at various points along the way.

Message-level security concerns in SOAP web services are addressed through a suite of security standards. The WS-* stack of specifications comprises many standards; the three security standards we'll review are WS-Security, WS-Trust, and WS-SecureConversation. We'll examine these standards in the context of a common integration pattern: a web services integration to an enterprise service bus. This pattern lets us put the standards through their paces to show how some message protection standards work in a distributed yet highly integrated system.

WS-Security

WS-Security [OASIS 2006b[21], Nadalin 2006[22]] is an open OASIS standard authored by IBM, Microsoft, and Verisign. The WS-Security standard describes how to associate XML Encryption and XML signature services with XML messages in web services. SSL is frequently deployed to deliver some protection to web services, but SSL's protection only extends to point to point scenarios and does not deliver message-level security. WS-Security lets service requesters and service providers protect their messages beyond the point to point horizon through a set of security services. WS-Security addresses message-level security through binding security tokens to XML messages. These security tokens represent claims made by or on behalf of a service requester and may be used by the service provider for message-level authentication, authorization, confidentiality, and integrity services. WS-Security is implemented on Microsoft, Java, Apache, and other technologies.

The WS-Security header is denoted in the XML message as

```
<wsse:Security>
...
</wsse:Security>
```

WS-Security allows the service requester or provider to encrypt and sign parts of a given message. This allows for a flexible integration where sensitive data may be encrypted and signed, but because message-level security is not an all or nothing proposition, the expense and complexity of these security mechanisms may be limited to specific message parts. The WS-Security header contains timestamp, encryption, digital signature, and security token data to provide message security services.

Timestamp

The timestamp is included in the header for the service provider to evaluate the length of time since the claims (for example, authentication claims) were made in the message and when the message is read by the service provider. In an asynchronous system such as an enterprise service bus or more elaborate SOA orchestrations, significant time may elapse between the time a message is generated by a service and the time it reaches the implementation consumer. One of the main uses of the WS-Security message timestamp is to introduce some entropy in the message to protect against replay attacks.

21. #dsy639-BSI_OASIS 2006b
22. #dsy639-BSI_Nadalin 2006

The timestamp also allows the service to stamp an expiration date on the message's claims so that the service provider knows to accept claims only within a given time parameter. For example, when authorizing a payment on a credit card, a payment system may hold a sum of money against a credit card for a period of time; if the transaction is not completed within the given time, a new authorization may need to be generated.

The timestamp is represented in the XML message in the WS-Security header:

```
<wsse:Security>
<wsu:Timestamp>
<wsu:Created >2006-08-09T06:12:03Z</wsu:Created>
<wsu:Expires >2006-08-09T08:12:03Z</wsu:Expires>
</wsu:Timestamp>
</wsse:Security></p>
```

This message expires after two hours. The timestamp may be targeted by attackers, since altering its contents may force the service provider to discard an otherwise legitimate message. Hence the timestamp should have integrity protection as well.

One implementation pattern used for efficiency (but not required by the WS-Security specification) is to place the <wsu:Timestamp> at the beginning of the header so that it may be evaluated first. This prevents time being wasted evaluating an expired message.

WS-Security token types

WS-Security headers may contain three different types of security tokens: username, binary, and XML tokens.

Username token

The username token is the most basic type of security token in WS-Security. The username token is a simple XML description of the username the service claims to represent. The basic username token is unsigned, making it a weak assurance option for protecting messages. The following listing shows an example usage of the username token:

```
<wsse:UsernameToken>
<wsse:Username>JillServiceUser</wsse:Username>
</wsse:UsernameToken>
```

The username token is made stronger by signing it as part of the message and by adding a password, either in the form of a plaintext password (which would be a poor choice for messages passed over any communications channel that is not highly secure, or arguably even over channels that are considered to be very secure) or as a password digest.

```
<wsse:UsernameToken>
<wsse:Username>JillServiceUser</wsse:Username>
<wsse:Password Type="...#PasswordDigest">
J45qK3rgHhFV8t3Rw==
</wsse:Password>

</wsse:UsernameToken>
```

Binary security tokens (X.509 and Kerberos)

X.509 digital certificates and Kerberos tickets are binary security tokens that are first encoded as binary and then represented in XML documents passed between the services. The tag <wsse:BinarySecurityToken> denotes these token types in the WS-Security headers. These token types allow for software security architects to integrate their existing identity and access management systems, such as PKI, LDAP, and Active Directory, into their SOA applications. This extends the reach of the identity and access management infrastructure and allows the software security to select from a variety of methods to propagate identity credentials such as impersonation, delegation, and attribution [Peterson 2005[23]].

23. #dsy639-BSI_Peterson 2005

Example X.509 certificate encoded in WS-Security header

```
<wsse:Security>
<X509Data>
<X509IssuerSerial>
<X509IssuerName>OU=HugeAirline,O=IBM,L=Central,ST=Oklahoma,C=US
</X509IssuerName>
<X509SerialNumber>1250</X509SerialNumber></X509IssuerSerial>
<X509SubjectName>CN=Jill Service User</X509SubjectName>
<X509Certificate>
BgTCE9rbGFob21hMRAwDgYDVQQHEwdVbmsam3duMQwwCgYDVQQKEwNJQk0xDTAM\
IIB0TCCAToCAQAwDQYJKoZIhvcNAQEEBQAwTzELMAkGA1UEBhMCVVMxETAPBgNV\
LBgNVBAsTBEphdmEwHhcNMDIwOTI1MTAxMTQ4WhcNMDMwOTI1MTAxMTQ4WjATMR\
EwDwYDVQQDEwhKb2huIERvZTCBnzANBgkqhkiG
...
</X509Certificate>
</X509Data>
</wsse:Security>
```

While X.509 and Kerberos can provide higher assurance than username tokens, they do add complexity to applications. The balance that the software security architect must seek is evaluating the number of systems that are to be integrated that already use security credentials from X.509 and Kerberos systems. When messages are routed throughout the system, other service providers may have constraints that do not allow them to consume and understand X.509 and Kerberos tokens. Historically, many systems integration patterns dealt with this issue with a lowest common denominator approach, effectively using username/passwords, hard coded values in code or config files, SSL point-to-point transport layer security, and other suboptimal (from a security point of view) workarounds. There are several ways to deal with this issue in WS-*. WS-Trust, described in the next section, provides a way to broker token types. Another alternative tactic is to target headers to specific services.

XML security tokens (SAML)

WS-Security and SAML [OASIS 2006[24]] both provide some similar solutions in web services security, and in some cases may be used instead of each other. WS-Security is able to leverage SAML as an XML security token type. SAML's security model uses *assertions* that are mediated between an assertion producer and assertion consumer, which is conceptually similar to what the WS-* model calls *claims*. SAML is bound to specific use cases, such as browser-based single sign on. SAML assertions have profiles and bindings for each use case. SAML assertions may be used as an XML security token representing authentication, authorization, and attribute statements. WS-Security provides the framework to bind SAML tokens to SOAP messages.

```
<wsse:Security xmlns:wsse="...">
<saml:Assertion xmlns:saml="..."
IssueInstant="2005-12-01T14:28:31.023Z">
<saml:Issuer>http://fooSAMLprovider/</saml:Issuer>
<saml:AuthenticationStatement>
<saml:Subject>
<saml:NameIdentifier
NameQualifier="exampledomain" >
uid=JillServiceUser,ou=HugeAirline ,o=IBM
</saml:NameIdentifier>
<saml:SubjectConfirmation>
<saml:ConfirmationMethod>
urn:oasis:names:tc:SAML:1.0:cm:bearer
</saml:ConfirmationMethod>
</saml:SubjectConfirmation>
</saml:Subject>
</saml:AuthenticationnStatement>
</saml:Assertion>
</wsse:Security>
```

24. #dsy639-BSI_OASIS 2006

For systems that are being built today, SAML tokens may be the most interoperable token format, based on the number of systems (from web applications to web services) that are able to consume SAML assertions. If there are no SAML profiles and bindings for systems that messages need to interoperate with, then other token types, configuration and coding become necessary. The WS-Trust standard (described in the next section) supports traversal across domains using multiple token types.

Targeting WS-Security headers for specific system capabilities

A SOAP message may have several WS-Security headers, each with its own token type targeted at specific SOAP actors or roles. This means that a Kerberos header can be targeted to systems using Active Directory, while systems that can only understand X.509 may have a header customized for their systems. Many view a central security solution as a holy grail, but in most organizations, these only exist on architects' white boards. In reality, especially in integration, the world is multicentered. The security solution must reflect this multicentered reality and offer solutions either at runtime or through other off-line mechanisms such as provisioning, so that the dreaded lowest common denominator approach does not lead to a security solution that is less strong than the sum of its parts.

WS-Security headers are targeted through the actor or role tag, so <wsse:Security S11:actor="FooWindowsServer"…> could target a specific header for a Windows system, and <wsse:Security S11:actor="BarMainframeSystem"...> could target a header for a mainframe system. One challenge in this pattern is that the Web Service Provider must deal with numerous token types, where it may not be able to negotiate all the token protocols at runtime. To traverse security domains, WS-Trust (described later) provides functionality to address these issues on the service provider side through token exchange, which represents a more robust pattern for some implementations because the logic is built into the service provider side instead of the service requester.

AJAX-style web services

AJAX-style applications use scripting languages such as JavaScript and VBScript on the client, because the scripting functionality provides a rich user interface and valuable user functionality. However, given the lack of security features in the scripting languages and the server's inability to enforce policy on the AJAX client, AJAX applications open up many new attack vectors, particularly through Cross Site Scripting (XSS) attacks. XSS attacks, when combined with the rapid, asynchronous communication that AJAX apps feature and the wide distribution of Web applications, create security vulnerabilities that cascade in unexpected ways. One example of an XSS attack in the wild is the MySpace worm, which infected over a million users in under 20 hours [Lai 2005[25]].

The tokens that WS-Security enabled messages describe may be used by the service provider to perform authentication, authorization, confidentiality, and integrity services at the message level.

Encryption and digital signatures

The WS-Security standard provides a description for how to associate security tokens and service with XML messages in a web services system. The core encryption and digital signature operations are actually performed by open standards XML Encryption [Imamura 2002[26]] and XML Signature [Bartel 2002[27]].

XML Encryption provides symmetric and asymmetric encryption capabilities for the XML message, so the message itself is protected end to end (not just within the communication channel as with SSL). XML Encryption may be used to encrypt the whole XML document, an XML element, or just an XML element's content. For example, assume that an XML document contains information about a traveler, and that the document contains an XML element named *PassportNumber*. The content of this XML element is a string of characters representing the traveler's passport number. Encrypting the XML element's content protects the number from being disclosed to unwelcome intermediaries, but encrypting the entire *PassportNumber* element helps prevent such intermediaries from even finding out that the document contains a passport number. The coarsest level of encryption granularity protects all of the information by encrypting the

26. #dsy639-BSI_Imamura 2002
27. #dsy639-BSI_Bartel 2002

whole XML document. However, this can prevent legitimate intermediate agents from making informed processing decisions based on some document attributes that could have been left in the clear if finer levels of encryption granularity had been used.

XML Signature may be used to sign all or part of a XML message. XML Signatures may be *enveloping* where the signature wraps the data, *enveloped* where the signature is wrapped by the XML, or *detached* where the signature is retrieved remotely, such as via a URI. XML Signature specifies the signature types and the canonicalization and transformations, which are particularly important in an XML based system.

In summary, WS-Security provides some important security services for web services and allows the development staff to get out of the business of having to code custom security mechanisms by hand. The WS-Security standard does not address other issues related to security infrastructure such as key management, and it does not address policy, which must be set up separately. How to establish and express security policies is beyond the scope of this article, but emerging standards include WS-Policy [Bajaj 2006[28]], which provides a framework for expressing policies in the form of policy assertions, and WS-SecurityPolicy [Della-Libera 2005[29]], which complements WS-Policy by providing a standard way of expressing policy assertions for security.

WS-Trust

Like WS-Security, WS-Trust is an open standard [Nadalin 2006b[30]]. WS-Security is concerned with associating security tokens with XML messages; WS-Trust is focused on how to make security tokens interoperable on a heterogeneous, integrated, networked system. WS-Trust calls are straightforward, functioning on a request-response message exchange pattern:

- Request Security Token (RST)

```
  <wst:RequestSecurityToken Context="...">
<wst:TokenType>...</wst:TokenType>
<wst:RequestType>
    ...
  </wst:RequestType>
</wst:RequestSecurityToken>
```

- Request Security Token Response (RSTR)

```
  <wst:RequestSecurityTokenResponse Context="...">
<wst:TokenType>...</wst:TokenType>
  <wst:RequestedSecurityToken>... </wst:RequestedSecurityToken>
  ...
</wst:RequestSecurityTokenResponse>
```

In WS-Trust, a Security Token Server (STS) is used to handle RST calls. The security tokens supported by WS-Trust are the same tokens supported by WS-Security: Username, X.509, Kerberos, and SAML. In addition, since the tokens are represented in XML, the message can contain proprietary and homegrown security tokens such as session cookies and mainframe tokens.

The STS provides four main functions:

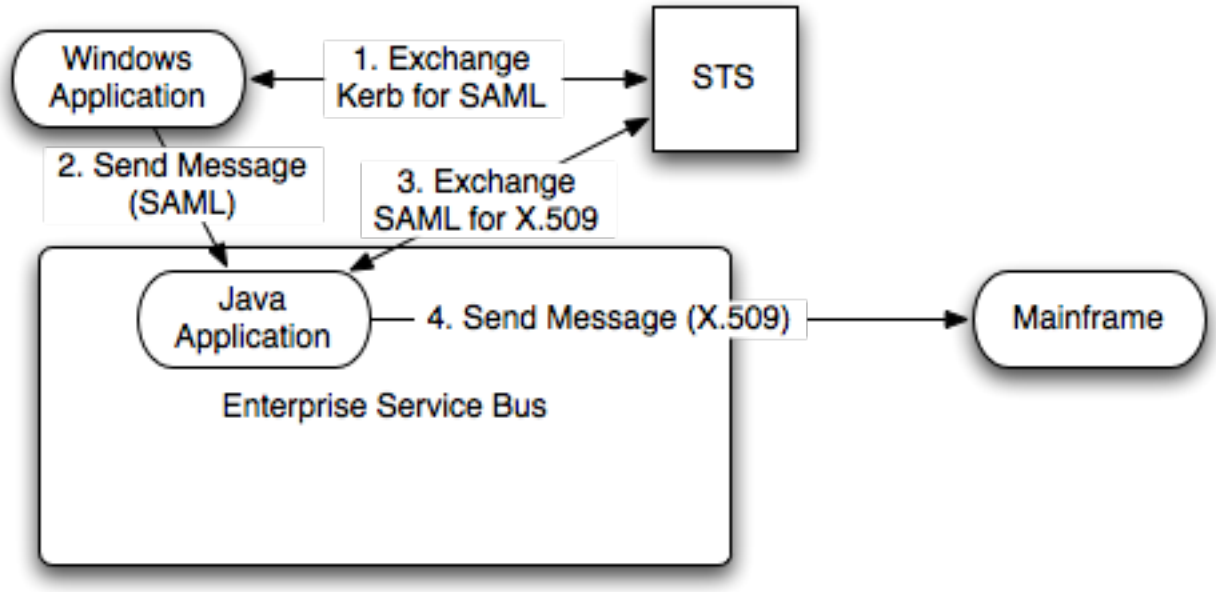
- validating security tokens
- issuing security tokens
- cancelling security tokens
- renewing security tokens

The combination of these functions effectively allows for exchanging security tokens, which is where the real power of WS-Trust is. The main focus of web services is interoperability, and interoperability is usually driven by business, not technical, concerns. Hence the systems that need to be integrated may run in

28. #dsy639-BSI_Bajaj 2006
29. #dsy639-BSI_Della-Libera 2005
30. #dsy639-BSI_Nadalin 2006b

completely different geographic, organizational, and technical paradigms. Imagine an enterprise service bus system that integrates Windows applications, which have Kerberos tickets, Java applications, which have SAML assertions, and mainframe applications, which have X.509 certificates. Assuming a business reason why all these systems must work together, what is the best way to get all these security tokens to compose?

Figure 6. Security token server (STS) providing exchange across token formats (Kerberos, SAML, and X.509)



In the diagram above,

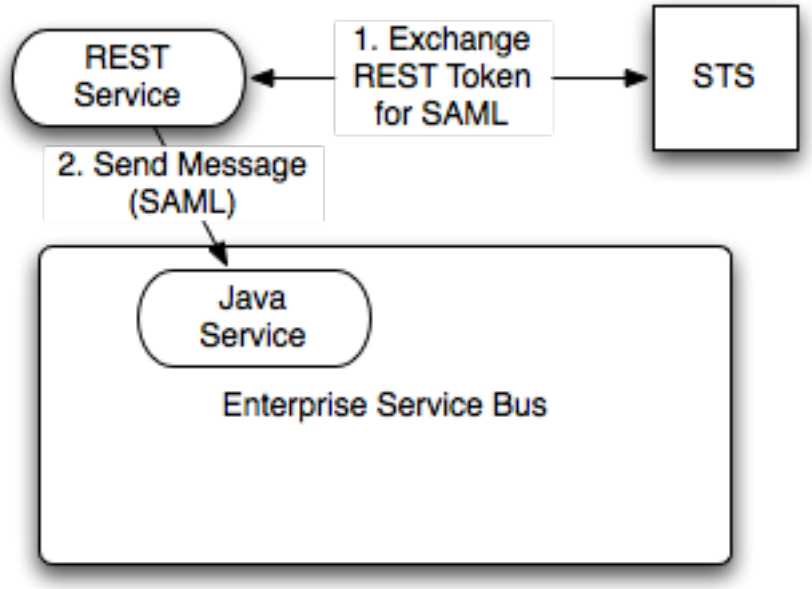
1. A Windows application uses WS-Trust to call the STS and exchange its Kerberos ticket for a SAML assertion.
2. The Windows application posts a Web Service message using a SAML token in a WS-Security header.
3. The Java application on the enterprise service bus exchanges the SAML assertion for a X.509 certificate.
4. The Java application sends a message to the mainframe using the X.509 certificate token.

(Not shown): Validation may occur at any step by having the STS query the system that produced the token. For example, in step 1 the STS may query Active Directory to validate the initial Kerberos ticket.

Depending on the security credential mapping, the software security architect may elect the STS to enforce a delegation, impersonation, attribution, or other method for proxying security tokens. Composition of security token types and symmetric and asymmetric authenticators make web services a more consistent way to integrate systems.

WS-Trust can be used to broker tokens from other systems as well—remember the fact that REST web services lack a security standard? Assume that a REST web service needs to get information from a SOAP web service based interface to an enterprise service bus. The STS can facilitate the exchange of the REST token for the security tokens in use on the enterprise service bus. Even though REST does not have a security standard, it is still in use on many systems. In this case, the software security architect need not resort to the lowest common denominator style of security. Instead, the system can be configured to take advantage of the tokens that the REST request offers through a service provider and use an STS to exchange credentials for further propagation. In the diagram below, the REST service exchanges its security token via the STS for a SAML assertion. The SAML assertion is sent by the service requester to the enterprise service bus's web service provider. The enterprise service bus may now propagate a SAML assertion to its service subscribers.

Figure 7. Security token server providing exchange between SAML and REST token formats



Since the STS functions like an identity middleware system, it provides a location for auditing service usage and enforcing policy across systems. For integration systems like an enterprise service bus, the ability to exchange tokens to integrate multiple security domains is particularly useful.

WS-Trust opens up new ways of approaching security when integrating systems through open standards. Its message-oriented approach, with security tokens and context attached to the message, allows for the security perimeter to be at the document level instead of at the (often massive) network level.

WS-SecureConversation

As we have described, WS-Security is focused on delivering security at the message level and WS-Trust is about extending security across systems for increased interoperability. Another security standard, WS-SecureConversation [Nadalin 2006c[31]], is focused on optimizing resource use. When using security services, there is a lot of computational expense to set up and tear down security contexts, and computational expense generally results in making business performance and risk management tradeoffs (vis-à-vis how much security weighed against execution time and resource cost). WS-SecureConversation allows multiple messages in web services to be exchanged using the same context. This alleviates the need to create a new security context with each message. For example, a signature may be checked to establish the context, and that context is set for either a period of time or an amount of messages.

The initialization process for WS-SecureConversation creates a SecurityContextToken (SCT), which may be created through WS-Trust. The SCT is passed with each subsequent message, as opposed to passing a normal security token with each message that must be independently checked. The lifespan for an SCT typically specifies a number of messages or a timespan. Optionally, SCTs may be renewable.

WS-SecureConversation builds upon the modular standards WS-Security and WS-Trust. WS-SecureConversation uses WS-Security token types, and SCTs may be bound to WS-Trust exchanges to optimize complex security token exchanges.

Similar to how connection pooling is done between an application server and a database server to minimize computational overhead, WS-SecureConversation is particularly useful in high-transaction environments, such as an enterprise service bus. The enterprise service bus may establish an SCT with subscribing systems and then use that SCT for a period of time. For example, in the ESB industry example described earlier in this paper, the system may establish an SCT for the travel agent for the length of a personnel shift.

31. #dsy639-BSI_Nadalin 2006c

Security Design Considerations in Web Services

Standards provide important authentication and message-level security services for web services, but they are not a complete security solution. In this section we also examine some additional areas for the software security architect to focus on in web services from a defense-in-depth standpoint.

Leverage Existing Standards

Start by leveraging existing standards. As mentioned above, these will not solve all of your security problems, but they remain the best starting point for several reasons. First, they are well reviewed by industry experts, next, they are updated regularly, and lastly, standards offer tooling opportunities so that optimizations may occur. For example XML Security Gateways may offer improved XML security services over what is available in custom coding.

Identity and Access Management in Web Services

Identity and access management (IAM) remains a challenge in distributed systems. WS-Trust and other standards offer new ways to solve identity problems at runtime, but these standards may not always be available, the organization may not have access to source code, and the system may use tokens that are not supported by WS-Trust. IAM systems constitute a number of different solutions that work off-line (like provisioning systems) and online to enforce policy and keep accounts' representations consistent. This can be particularly complex in web services because there may be different technologies in use, different namespaces, and synchronization and replication may leave digital identity detritus.

Heterogeneity Can Improve Security and Survivability

Lack of diversity is a well-understood risk factor for security. One security benefit from web services is that its wide support of platforms means that a given application may use many different technologies, which may reduce the impact of an attack that exploits a specific vulnerability in a particular software or hardware platform. A heterogeneous system can be more resilient in the face of viruses and other malware if it is designed to provide diverse and redundant means for delivering key services. Parts of the system may be affected by an attack but, if well-architected for survivability [Lipson 1999[32]], the system as a whole might still provide essential services. An enterprise service bus or other asynchronous messaging system may further protect against cascade failure by providing an abstraction layer that could prevent attacks from accessing the underlying technologies.

Input Validation Is Still Critical

As helpful as the web services standards are, there is no standard for input validation, and so many of the most nefarious attacks remain a threat. SQL injection, LDAP injection, and XPath injection all rely on inadequate input validation on the service side. These security problems remain just as they have with all the distributed system paradigms that preceded web services.

Sure It's a "Standard"—But Your Mileage May Vary

Standards are one thing. Executable code is another. Tools may offer different support for standards, so a system that advertises WS-Security support may not offer all token types. Many of the first generation Security Token Service (STS) products take in all four token types but only emit SAML. This is a fact of life with standards, especially in the early days, so checking in advance for tool support is a wise thing to do. Moreover, in an environment of emerging standards there is some concern about having "too many standards," or at least too much complexity in the current versions of these standards.

32. #dsy639-BSI_Lipson 1999

You’re in Uncharted Territory, So Carefully Explore Scenarios of Use and Misuse

Some aspects of web services security may seem similar to well-known security scenarios, but in many cases you’ll find yourself on unfamiliar ground. For example, message-level security may, on the surface, appear to be quite similar to protecting email messages in transit through the use of PGP (or GPG) encryption and digital signatures. However, unlike email, web services messages may need to be read, processed, and modified en route by legitimate intermediaries (e.g., service providers), yielding a far more complex security scenario than simply protecting a message between two endpoints. Since Web service security may quickly take you into uncharted territory, it is particularly important to thoroughly explore scenarios of use and misuse of proposed web services early in the development life cycle and use the results to help you specify your security requirements. Waiting to discover security problems in the field that could have been discerned and addressed before deployment is a recipe for disaster.

Service Assurance Is Required

Even if your messages are 100% secure in transit, many of the services you are dependent on are not in your direct control, and neither is their security. Even for services entirely in your control, there is still the real possibility of insider threat. It’s therefore critical to establish mechanisms for detecting security violations through auditing and/or through network and host intrusion detection. Unfortunately, there is currently no standards support for web services in this area.

Still No Audit Standard

There is no standard for auditing of web services, and so these systems must be developed individually. Additionally there is not an “out of the box” way in most web services implementations to correlate service requests and responses—so if this is required by your audit standards this is also a custom development project. There are some features in the WS-Addressing standard (such as separate ReplyTo addresses) that can be cleverly used in an ad hoc manner to implement an audit trail, but these features are not part of any audit standard.

Still No Standards that Aid Detection of Security Violations

Web services and WS-* offer an array of protection services, but detection services remain a custom effort. Depending on implementation, the network intrusion detection (NIDS) and host intrusion detection (HIDS) in your environment may or may not be able to analyze traffic. With the movement toward web services, HIDS are increasingly important because the services lack knowledge over what systems are connecting them and hence should be hardened to a DMZ or better level.

Security Policies Are Your Web Services Security Foundation

Web services standards do not create effective policy (though WS-Policy and WS-SecurityPolicy are used to express policy)—the creation and coordination of security policies are the responsibility and obligation of the participating organizations. Security policies are the starting point for establishing a set of security requirements that will serve as the foundation (supported by the WS-* standards) for building security into web services applications. A collaborative review of their security policies by participating organizations can help to resolve incompatibilities among the various policies (and the corresponding sets of security requirements derived from those policies). Moreover, the collaborative review can itself help to engender and improve trust in the systems and services controlled by those participating organizations.

Security policies should not only specify what systems and services are supposed to do (and what they should never do) but also should specify what actions must be taken when things go wrong. Good security practice mandates the creation and periodic review of policies and procedures for reporting, response, and recovery based on the discovery of security violations or other security problems.

Conclusion

Web services security is still a work in progress, both from the development of emerging standards and from the security community's growing awareness of the benefits, traps, and pitfalls of this new integration paradigm. With its primary focus on document-level integration and virtualization, the web services approach abstracts away the details of underlying platforms and operating systems, which reduces system coupling and much of the tight coordination and synchronization required by traditional integration methods. However, because of the complexity of coordinating (and achieving compatibility) of security policies and the security requirements derived from those policies (as reflected, for example, in a common message schema) across all of the participants in a web services application, it is more important than ever to build security into the cooperating systems and services early and throughout the software development life cycle.

Moreover, although message-level security is a primary concern in web services, protecting messages in transit does not obviate the need for following good security practices at the network, host, application, personnel (manual processes), and physical layers. Best security practices, which are common to (and essential for) all systems and integration patterns for both development and operations, can be found throughout the BSI website.

Finally, the web services approach typically involves interoperability with many services that are not in your organization's direct control, and limited ability to provide assurance proof. Therefore, monitoring web services operations for security violations (through auditing and the use of intrusion detection tools) is an essential aspect of defense in depth. However, there are presently no web services standards for audit or detection, so it's up to your own organization (in collaboration with other participating organizations) to build this critical capability into your systems from the outset, along with policies and procedures for recovery and response when problems are detected.

Bibliography

[Bajaj 2006]	Bajaj, Siddharth, et al. *Web Services Policy 1.2—Framework (WS-Policy)*[33]. W3C Member Submission, April 25, 2006.
[Bartel 2002]	Bartel, Mark; Boyer, John; Fox, Barb; LaMacchia, Brian; & Simon, Ed. Edited by Eastlake, Donald; Reagle, Joseph; & Solo, David. *XML-Signature Syntax and Processing*[34]. W3C Recommendation, February 12, 2002.
[Chappell 2004]	Chappell, David A. "Enterprise Service Bus." O'Reilly & Associates, 2004.
[Chinnici 2006]	Chinnici, Roberto; Moreau, Jean-Jacques; Ryman, Arthur; & Weerawarana, Sanjiva; eds. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*[35]. W3C Candidate Recommendation, March 27, 2006.
[Della-Libera 2005]	Della-Libera, Giovanni, et al. Web Services Security Policy Language (WS-SecurityPolicy), Version 1.1[36]. International Business Machines Corporation, Microsoft Corporation, RSA Security Inc., and VeriSign Inc., July 2005.
[Fielding 2000]	Fielding, Roy Thomas. "Architectural Styles and the Design of Network-Based Software Architectures[37]." PhD Dissertation, University of California, Irvine, 2000.

[Gudgin 2003] Gudgin, Martin; Hadley, Marc; Mendelsohn, Noah; Moreau, Jean-Jacques; & Nielsen, Henrik Frystyk; eds. *SOAP Version 1.2 Part 1: Messaging Framework*[38]. W3C Recommendation, June 24, 2003.

[Hohpe 2004] Hohpe, Gregor & Woolf, Bobby. *Enterprise Integration Patterns*. Boston, MA: Addison-Wesley Professional, 2004.

[IBM 2002] IBM Corporation & Microsoft Corporation. *Security in a Web Services World: A Proposed Architecture and Roadmap*[39]. Joint Security Whitepaper (Version 1.0), April 7, 2002.

[Imamura 2002] Imamura, Takeshi; Dillaway, Blair; & Simon, Ed. Edited by Eastlake, Donald & Reagle, Joseph. *XML Encryption Syntax and Processing*[40]. W3C Recommendation, December 10, 2002.

[Iwasa 2004] Iwasa, Kazunori (principal editor); Durand, Jacques; Rutt, Tom; Peel, Mark; Kunisetty, Sunil; & Bunting, Doug (assisting editors). *WS-Reliability 1.1*[41]. OASIS Standard Specification, November 15, 2004.

[Krafzig 2005] Krafzig, Dirk; Banke, Karl; & Slama, Dirk. *Enterprise SOA Service Oriented Architecture Best Practices*. Upper Saddle River, NJ: Prentice Hall PTR, 2005.

[Lai 2005] Lai, Eric. "Teen Uses Worm to Boost Ratings on MySpace.com[42]." *Computerworld*, October 17, 2005.

[Lipson 1999] Lipson, Howard & Fisher, David. "Survivability —A New Technical and Business Perspective on Security[43]," 33–39. *Proceedings of the 1999 New Security Paradigms Workshop*. Caledon Hills, Ontario, Canada, Sept. 22–24, 1999. New York: Association for Computing Machinery, 2000.

[Lipson 2005] Lipson, Howard & van Wyk, Ken. "Application Firewalls—Introduction and Concept of Operations[44]." Carnegie Mellon University, 2005.

[Nadalin 2006] Nadalin, Anthony; Kaler, Chris; Monzillo, Ronald; & Hallam-Baker, Phillip; eds. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*[45]. OASIS Standard Specification, February 1, 2006.

[Nadalin 2006b] Nadalin, Anthony; Goodner, Marc; Gudgin, Martin; Barbir, Abbie; & Granqvist, Hans; eds. *WS-Trust 1.3*[46]. Committee Draft 01, OASIS, September 6, 2006.

[Nadalin 2006c] Nadalin, Anthony; Goodner, Marc; Gudgin, Martin; Barbir, Abbie; & Granqvist, Hans; eds. *WS-*

SecureConversation 1.3[47]. Committee Draft 01, OASIS, September 6, 2006.

[Newcomer 2004] Newcomer, Eric & Lomow, Greg. *Understanding SOA with Web Services*. Boston, MA: Addison-Wesley, 2004.

[OASIS 2006] OASIS Security Services (SAML) Technical Committee[48].

[OASIS 2006b] OASIS Web Services Security (WSS) Technical Committee[49].

[OASIS 2006c] OASIS. OASIS Universal Description, Discovery and Integration (UDDI) Specification Technical Committee (UDDI Spec TC)[50].

[Peterson 2005] Peterson, Gunnar. "Identity in Assembly and Integration[51]." Cigital, 2005.

[Sessions 2004] Sessions, Roger. "Fuzzy Boundaries: Objects, Components, and Web Services[52]." *ACM Queue 2*, 9 (Dec/Jan 2004-2005).

[Shirey 1994] Shirey, Robert W. *Security Architecture for Internet Protocols: A Guide for Protocol Designs and Standards*. Internet Draft: draft-irtf-psrg-secarch-sect1-00[53], November 1994.

[Sun 2006] Sun Microsystems. Sun's XACML Implementation[54] (2006).

[W3C 2006] W3C. XML Protocol Working Group (SOAP)[55].

[W3C 2006b] W3C. Extensible Markup Language (XML)—Working Groups and Resources Web Page[56].

Carnegie Mellon Copyright

1. mailto:permission@sei.cmu.edu

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com